(19) JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: **2002229445 A**

(43) Date of publication of application: **14.08.02**

(51) Int. Cl **G09C 1/00**
G06F 7/72

(21) Application number: **2001021128**

(22) Date of filing: **30.01.01**

(71) Applicant: **MITSUBISHI ELECTRIC CORPMITSUBISHI ELECTRIC SYSTEM LSI DESIGN CORP**

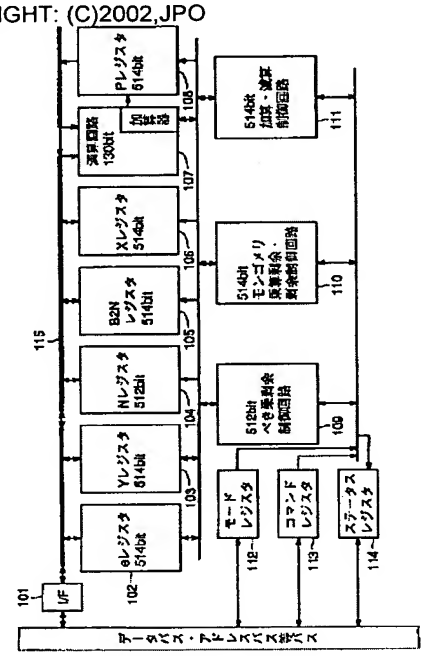
(72) Inventor: **ASAMI KAZUO**

(54) MODULATOR EXPONENT DEVICE

(57) Abstract:

PROBLEM TO BE SOLVED: To provide a modulator exponent device which can perform high-speed processing.

SOLUTION: The modulator exponent circuit includes an I/F (interface) circuit 101 which is an interface to an external bus, an e register 102 which holds a key e, a Y register 103 which holds a multiplier Y for making Montgomery transformation, an N register 104 which holds a key N, a B2N register 105 which holds the value of 2B+N to be performed in computation of the Montgomery transformation, an X register 106 which holds plaintext X, an arithmetic circuit 107 for making computation for the purpose of encryption and decryption, a P register 108 which holds the result P of the computation, a power-residue control circuit 109 which is ought to play the role as a state machine in executing the modulator exponent, a Montgomery multiplication residue and residue control circuit 110 which plays the role as a state machine in executing Montgomery modulator exponent and remainder computation and an addition and subtraction control circuit 111 which performs the computation control of addition and subtraction.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－229445
（P2002－229445A）
(43)公開日　平成14年８月14日(2002.8.14)

(51)Int.Cl.'	識別記号	ＦＩ	テーマコード'(参考)
Ｇ０９Ｃ　1/00	６２０	Ｇ０９Ｃ　1/00	６２０Ａ　５Ｊ１０４
Ｇ０６Ｆ　7/72		Ｇ０６Ｆ　7/72	

審査請求　未請求　請求項の数６　ＯＬ　（全 10 頁）

(21)出願番号　特願2001－21128(P2001－21128)

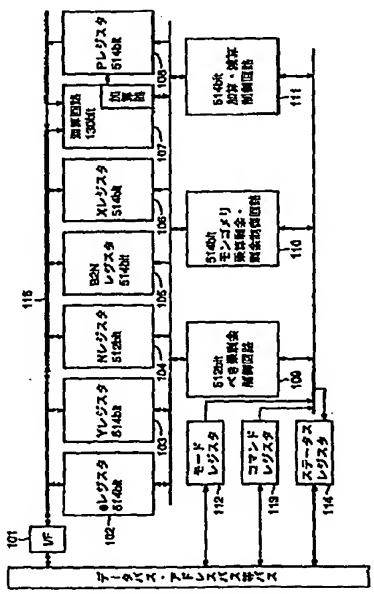
(22)出願日　平成13年１月30日(2001.1.30)

(71)出願人　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目２番３号
(71)出願人　391024515
三菱電機システムエル・エス・アイ・デザイン株式会社
兵庫県伊丹市中央３丁目１番17号
(72)発明者　朝見　和生
兵庫県伊丹市中央３丁目１番17号　三菱電機システムエル・エス・アイ・デザイン株式会社内
(74)代理人　100064746
弁理士　深見　久郎　（外４名）

Ｆターム(参考)　5J104 AA22 JA28 NA18

(54)【発明の名称】　べき乗剰余演算器

(57)【要約】
【課題】　高速処理が可能なべき乗剰余演算器を提供する。
【解決手段】　べき乗剰余演算回路は、外部バスとのインタフェースであるＩ／Ｆ（インタフェース）回路１０１と、鍵ｅを保持するｅレジスタ１０２と、モンゴメリ変換をする乗数Ｙを保持するＹレジスタ１０３と、鍵Ｎを保持するＮレジスタ１０４と、モンゴメリ変換の演算時に行なう２Ｂ＋Ｎの値を保持するＢ２Ｎレジスタ１０５と、平文Ｘを保持するＸレジスタ１０６と、暗号化および復号化のための演算を行なう演算回路１０７と、演算結果Ｐを保持するＰレジスタ１０８と、べき乗剰余演算実行時のステートマシンとしての役割を果たすべき乗剰余制御回路１０９と、モンゴメリ乗算剰余演算と剰余演算との実行時のステートマシンとしての役割を果たすモンゴメリ乗算剰余・剰余制御回路１１０と、加算および減算の演算制御を行なう加算・減算制御回路１１１とを含む。



1

【特許請求の範囲】

【請求項１】　モンゴメリ乗算剰余演算を行なう際の一方の引数を２倍した値と剰余の法とを加算した値を保持するレジスタと、

前記レジスタに接続され、前記レジスタに保持された値を参照して、モンゴメリ乗算剰余演算を実行するためのモンゴメリ乗算剰余演算実行手段と、

前記モンゴメリ乗算剰余演算実行手段に接続され、前記モンゴメリ乗算剰余演算実行手段との間でデータのやり取りを行ない、べき乗剰余演算を実行するためのべき乗剰余演算実行手段とを含む、べき乗剰余演算器。

【請求項２】　前記べき乗剰余演算実行手段は、２進数表現されたべき指数の各ビットの値に関わらず、モンゴメリ乗算剰余演算実行手段においてモンゴメリ乗算剰余演算を実行し、べき乗剰余演算を実行する、請求項１に記載のべき乗剰余演算器。

【請求項３】　さらに、べき乗剰余演算実行手段における演算モードを保持するためのモードレジスタを含み、

前記べき乗剰余演算実行手段は、前記モードレジスタに保持された値に基づいて、２進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する、請求項１または２に記載のべき乗剰余演算器。

【請求項４】　前記べき乗剰余演算実行手段は、２進数表現されたべき指数の各ビットの値に基づいて、モンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する、請求項１～３のいずれかに記載のべき乗剰余演算器。

【請求項５】　前記べき乗剰余演算実行手段は、２進数のビット列を加算する加算器を含み、

前記加算器は，２進数のビット列を所定ビットごとに分割し、分割後のビット列同士で加算を行なう複数のサブ加算器を含む、請求項１～４のいずれかに記載のべき乗剰余演算器。

【請求項６】　さらに、前記モンゴメリ乗算剰余演算実行手段および前記べき乗剰余演算実行手段に接続され、演算の一部を取出して実行するための手段を含む、請求項１～５のいずれかに記載のべき乗剰余演算器。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、情報通信ネットワーク、交通、金融、医療、流通等の分野において使用される情報の暗号化技術および復号化技術で使用されるべき乗剰余演算器に関し、特に、モンゴメリ（Montgomery）のアルゴリズムを用いてべき乗剰余演算を行なうべき乗剰余演算器に関する。

【０００２】

【従来の技術】情報通信技術の発展に伴い、情報ネットワーク上のセキュリティの確保（データの盗用や破壊を＊

2

＊防止すること）が重要視されるようになってきている。そのため、情報の暗号化技術および復号化技術が採用されることが多く、その適用分野は単なる情報通信分野に留まらず、交通、金融、医療、流通等の身近な分野にまで広がりつつある。この種の暗号化技術および復号化技術には、高度なセキュリティを単純な原理によって実現できることが要求される。

【０００３】まず、この種の技術の理解を容易にするため、情報の暗号化・復号化についての概略を説明する。暗号の世界においては、“非対称暗号アルゴリズム”が質的に優れている。非対称暗号アルゴリズムとは、暗号化鍵と復号化鍵とが異なっており、そのいずれか一方から他方が“容易に計算できない”暗号アルゴリズムをいう。この非対称暗号アルゴリズムの代表的なものに、べき乗剰余演算（ある数Ｘを何回も乗算してＮで割った余りをとる計算）を用いるタイプのＲＳＡ（Rivest-Shamir-Adleman scheme）暗号がある。

【０００４】ＲＳＡ暗号を生成するには、次式（１）のべき乗剰余演算の形式が基本として使用される。式（１）は、X^YをＮで割ったときの余りを求めることを意味する。また、式（１）において、Ｘは暗号化（復号化）の対象となる平文、ＹおよびＮは暗号化（復号化）のための鍵（キー）である。

【０００５】$X^Y \bmod N$　　…（１）

このべき乗剰余演算を用いることにより、情報の暗号化および復号化が容易に実行され、かつＸ、Ｙ、Ｎのオペランドビット長を長くすることで、各鍵の解読を困難にすることができる。

【０００６】しかし、オペランドビット長を長くすると、べき乗剰余演算に長時間を要することになる。そこで、オペランドビット長が長いべき乗剰余演算をいかに短時間に終了させるかがポイントとなる。

【０００７】次に、ＲＳＡ暗号を例に取り、べき乗剰余演算を使用した暗号化処理および復号化処理について説明する。

【０００８】［ＲＳＡ暗号の暗号化および復号化について］

（１）　ＲＳＡ暗号の暗号化には、次式（２）が用いられる。

【０００９】$C = M^e \bmod N$　　…（２）

復号化には、次式（３）が用いられる。

【００１０】$M = C^d \bmod N$　　…（３）

ここで、Ｍは暗号化の対象となる平文、Ｃは暗号化された平文すなわち暗号文である。また、式（２）におけるｅおよびＮは暗号化鍵、式（３）におけるｄおよびＮは復号化鍵である。また、以下の式（４）および式（５）の関係が予め与えられている。

【００１１】

$$N = p \cdot q \quad \cdots (4)$$

$$1 \equiv e \cdot d \bmod \{LCM(p-1, q-1)\} \quad \cdots(5)$$

ここで、「≡」は、左辺と右辺とが相似であることを意味し、「LCM」は、最小公倍数を意味する。またpとqとは互いに素な整数である。なお、eおよびNは公開鍵であり、d、pおよびqは秘密鍵である。

【0012】式（4）および式（5）は、ともに暗号アルゴリズムにおけるべき乗剰余演算の数値の条件を定義している。式（4）は、Nは互いに素な大きな素数pおよびqの積であることを示している。pおよびqはともに奇数なので、当然Nは奇数でなければならない。式（5）は、式（4）で示したpおよびqからそれぞれ1を減じた値同士の最小公倍数で、eおよびdの積e・dを当該最小公倍数で割ったときの余りが1になることを示している。

【0013】式（4）および式（5）の条件に基づき、平文Mは式（2）を用いて暗号化され、また暗号化された平文M（暗号文C）は式（3）を用いて復号化される。

【0014】［べき乗剰余演算の演算方法について］次に暗号化・復号化で使用される、べき乗剰余演算の演算方法を説明する。$A = M^e \bmod N$のべき乗剰余演算は、整数eの2進数展開を$e = e^{k-1} \cdots e^1 e^0$として、以下のフロー1に示す反復平方積法を用いることにより実行される。

【0015】

（フロー1）

```
begin
  A=1
  for  i=k-1  to  0
    begin
      A=A²modN        …(6)
      If  eⁱ=1  then  A=A・MmodN      …(7)
    end
end
```

Aに格納された値が求めたいべき乗剰余演算の解になる。

【0016】以上のように演算の基本は、式（6）および式（7）に示すように乗算と除算（mod算）である。乗算は、初期値を1とするAの値に対してA×AまたはA×Mを行なう部分である。除算は、各々の乗算で得られた値に対してmodN（Nで割ったときの余りを求める演算）を行なう部分である。この“乗算と除算”（A×AmodN、A×MmodN）を1対の演算として、“e”のビット値に従って繰返し演算が行なわれる。すなわち、“e”の最上位ビットから最下位ビットまでの各ビットの内容によって“乗算と除算”が行なわれる。

【0017】べき乗剰余演算は、基本となる剰余演算（mod算）を繰返し行なうことで解が得られることを示したが、この繰返し回数自体は、たかだか数百～数千回であるので、ソフトウェアによる処理でも十分に対応できる。しかし、この剰余演算自体すなわち除算をハードウェアによって実行するためには、大規模な演算回路と複雑な処理手順とが必要とされる。このため改善が望まれていた。通常、e，d，M，Nなどは1024ビット程度の大きな整数が用いられているので、高速指数計算法を使用しても1回のRSA演算で平均1500回程度の多重精度乗算と剰余算とを行なわなければならない。特に剰余算は、近似法、剰余テーブル方式、モンゴメリのアルゴリズム等、多くの高速化手法が提案されている。

【0018】このような、RSA暗号に代表される公開鍵暗号の多くで利用される、べき乗剰余演算を高速に処理するためには、1回当りの剰余算の高速化が要求される。モンゴメリのアルゴリズムは、剰余算を高速処理するアルゴリズムである。特に、乗算剰余演算においては、除算をビットシフトなどで簡略化できるため、公開鍵暗号（RSA暗号等）で用いられるべき乗剰余演算を高速処理することができるという特徴がある。

【0019】一方、中国人の剰余定理により、合成数を法とする演算は合成数を構成する互いに素な因数を法とする演算から計算できる。これを1024ビット長RSA暗号処理に適用すると、実際に必要なハードウェアとしては、1024ビット長の法Nによるべき乗剰余演算回路ではなく、512ビット長の整数（ここではpおよびqに相当する）を法とする演算回路のみでよい。このためハードウェアの小型化に繋がる。

【0020】べき乗剰余演算は、基本となる剰余演算（mod算）を実行する手順が非常に複雑であるため、演算回路が大規模化してしまうことを上述した。そこで、モンゴメリは、剰余演算（mod算）を先のような一般的な方法で行なわずに、“乗算”と簡単なビット列処理とで行なうことによって解を得る仕組みを提案している。以下にモンゴメリが提案している手法について簡単に説明する。

【0021】［モンゴメリのアルゴリズム］剰余演算の高速化を実現する一手法であるモンゴメリのアルゴリズムについて説明する。

【0022】モンゴメリのアルゴリズムは、剰余の法N（N>1）と、剰余の法Nと互いに素である基数R（R>N）とを用いると、被剰余数をTとした場合、TR^{-1}

modNの計算が基数Rによる除算のみで行なえる性質を利用している。これにより、Nによる除算を用いることなく剰余計算を行なうことができる。ここで、N，R，R^{-1}およびTは整数である。被剰余数Tは$0 \leqq T < R \cdot N$を満たす数である。R^{-1}は剰余の法Nの上での基数Rの逆数である。またここでさらに、$R \cdot R^{-1} - N \cdot N' = 1$（$0 \leqq R^{-1} < N$、$0 \leqq N' < R$）の関係を満たす整数N′を考えることができる。さらに、この基数Rに2のべき乗数を使用した場合、基数Rによる除算をシフト操作に置き換えることができる。このため、$T \rightarrow TR^{-1} \bmod N$（被剰余数をTとした場合の$TR^{-1} \bmod N$）の計算の高速処理が可能となる。

【0023】次にアルゴリズム1として、$T \rightarrow TR^{-1} \bmod N$のアルゴリズムMR（T）を示す。ただし、アルゴリズム1において（T+m・N）／Rは必ず割り切れることが証明されている。

【0024】（アルゴリズム1）$T \rightarrow TR^{-1} \bmod N$のアルゴリズムY＝MR（T）は次のように表わされる。

【0025】

$$M = (T \bmod R) \cdot N' \bmod R \quad \cdots(8)$$

$$Y = (T + m \cdot N) / R \quad \cdots(9)$$

if Y≧N then Y＝Y−N

Y<N then return Y

1回のMRでは、剰余TmodNではなく$TR^{-1} \bmod N$が求められるだけである。よって、剰余TmodNを求めるためには、次に示すようにMR（T）と予め求めておいた$R^2 \bmod N$との積で、再びMR演算を行なえばよい。

【0026】

$$MR(MR(T) \cdot (R^2 \bmod N))$$
$$= (TR^{-1} \bmod N) \cdot (R^2 \bmod N) \cdot R^{-1} \bmod N$$
$$= TR^{-1} \cdot R^2 \cdot R^{-1} \bmod N$$
$$= T \bmod N$$

このようにして剰余TmodNを求めることができる。

【0027】以上のことにより、モンゴメリ法による乗算剰余演算を使用して、これをべき乗剰余演算の反復平方積法（繰返し2乗法）で実現するアルゴリズムを下記に示す。鍵eの上位ビットから検索し、鍵のビットの値が1の場合には、MR（X・Y）のモンゴメリ乗算剰余演算を行なう。

【0028】

$Y = Rr$ （$Rr = R^2 \bmod N$（$R = 2^{k+2}$））
X＝M
X＝MR（X，Y） …（10）
Y＝MR（1・Y） …（11）
for j＝k to 1
　if ej＝＝1 then Y＝MR（X・Y） …（12）
　if j＞1 then Y＝MR（X・Y） …（13）
end for
Y＝MR（1・Y） …（14）
Y＝YmodN …（15）

ここで、MR（X・Y）とMR（Y・X）とは等しく、ejはキーeのjビット目を表わす。512ビット長の整数の場合k＝512となり、512ビットのべき乗剰余演算は514ビットのモンゴメリ乗算剰余演算と512ビット剰余演算とにより実現できる。

$W = 2^d$
$N0' = N' \bmod W$
P＝0
for j＝0 to k
　M＝（PmodW）・N0′ …（16）
　$P = ((P + (A \bmod W) \cdot B \cdot W + M \cdot N) / W) \bmod 2^k$ …（17）
　A＝A／W …（18）
End

dは自然数であり、ハードウェアに依存する数である。このようにして、モンゴメリ乗算剰余演算結果Pを求めることができる。d＝1の基数2の逐次計算で514ビットモンゴメリ乗算剰余演算結果P＝MR（B・A）を求めると、以下のようになる。

【0029】また、ハードウェアとして実装するのに最適な基数Wの逐次計算でモンゴメリ乗算剰余演算結果P＝MR（B・A）を求めると、以下のようになる。

【0030】

$N0' = N' \bmod 2$
P＝0
for j＝0 to 514

【0031】

$$M=(P \bmod 2)\cdot N0' \quad \cdots(19)$$

$$P=((P+(A \bmod 2)\cdot B\cdot 2+M\cdot N)/2) \bmod 2^{514} \quad \cdots(20)$$

$$A=A/2 \quad \cdots(21)$$

end

以上のように、べき乗剰余演算を実現するために、ハードウェアで５１２ビット長のべき乗剰余演算にモンゴメリ法を使用し、ソフトウェアで中国人の剰余定理を利用した処理を使用することが従来採用されている。ハードウェアへの実装方式は複数通りあり、実際に様々な方式が採用されていると思われる。

【００３２】

【発明が解決しようとする課題】しかし、従来の回路では、図８に示すような手順で処理が行なわれている。すなわち、５１２ビット長のべき乗剰余演算にモンゴメリ法を使用した回路を実装したハードウェアを用い、式（１０）～式（１８）を式そのままに実行している。たとえば、ｅｊ＝＝０のときは式（１２）をスキップしている。一方、式（１７）においては毎回すべての計算を実行している。そのため複雑な処理手順が必要とされ、より一層の高速化が望まれている。また、回路規模が小さくて済み、ＬＳＩ（Large Scale Integration）化に適した回路が必要となっているので、演算処理をできるだけ簡略化し全体の計算量を削減し、処理速度の向上を図ることが必要である。

【００３３】本発明は上述の課題を解決するためになされたもので、その目的は、高速処理が可能なべき乗剰余演算器を提供することである。

【００３４】

【課題を解決するための手段】本発明のある局面に従うべき乗剰余演算器は、モンゴメリ乗算剰余演算を行なう際の一方の引数を２倍した値と剰余の法とを加算した値を保持するレジスタと、レジスタに接続され、レジスタに保持された値を参照して、モンゴメリ乗算剰余演算を実行するためのモンゴメリ乗算剰余演算実行手段と、モンゴメリ乗算剰余演算実行手段に接続され、モンゴメリ乗算剰余演算実行手段との間でデータのやり取りを行ない、べき乗剰余演算を実行するためのべき乗剰余演算実行手段とを含む。

【００３５】モンゴメリ乗算剰余演算に頻繁に用いられる値をレジスタに保持することにより、モンゴメリ乗算剰余演算を高速に実行することができる。

【００３６】好ましくは、べき乗剰余演算実行手段は、２進数表現されたべき指数の各ビットの値に関わらず、モンゴメリ乗算剰余演算実行手段においてモンゴメリ乗算剰余演算を実行し、べき乗剰余演算を実行する。

【００３７】べき指数の各ビットの値に関わらず、常にモンゴメリ乗算剰余演算が実行される。このため、べき乗剰余演算器を暗号化装置および復号化装置に用いた場合であっても、タイミング攻撃に対する耐性を確保することができる。

【００３８】さらに好ましくは、べき乗剰余演算器は、さらに、べき乗剰余演算実行手段における演算モードを保持するためのモードレジスタを含み、べき乗剰余演算実行手段は、モードレジスタに保持された値に基づいて、２進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する。

【００３９】モードレジスタに保持された値に基づいて、２進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するか否かを判断する。このため、テスト時には、２進数表現されたべき指数の各ビットの値に基づいたモンゴメリ乗算剰余演算を実行するようにし、実際使用する際には、べき指数の各ビットの値に関わらず、常にモンゴメリ乗算剰余演算を実行するようにすれば、テスト時間を短縮でき、かつタイミング攻撃に対する耐性を確保することができる。

【００４０】さらに好ましくは、べき乗剰余演算実行手段は、２進数表現されたべき指数の各ビットの値に基づいて、モンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する。

【００４１】２進数表現されたたべき指数の各ビットの値に基づいて、モンゴメリ乗算剰余演算を実行するか否かを判断し、モンゴメリ乗算剰余演算を実行する。このため、ビットの値として１が出現するまでの間は、モンゴメリ乗算剰余演算の結果が予めわかっている場合がある。このような場合に、処理をスキップすることにより、高速に処理することができる。

【００４２】さらに好ましくは、べき乗剰余演算実行手段は、２進数のビット列を加算する加算器を含み、加算器は、２進数のビット列を所定ビットごとに分割し、分割後のビット列同士で加算を行なう複数のサブ加算器を含む。

【００４３】加算器をサブ加算器に分割してパイプライン処理を実行することにより、高速に加算を行なうことができる。このため、べき乗剰余演算を高速に実行することが可能になる。

【００４４】さらに好ましくは、べき乗剰余演算器は、さらに、モンゴメリ乗算剰余演算実行手段およびべき乗剰余演算実行手段に接続され、演算の一部を取出して実行するための手段を含む。

【００４５】べき乗剰余演算時に行なう各種演算を取出して実行することにより、様々な種類の暗号化処理を実現できるようになる。

【００４６】

【発明の実施の形態】［実施の形態１］
［べき乗剰余演算回路の全体構成］図１を参照して、本発明の実施の形態１に係るべき乗剰余演算回路は、外部バスとのインタフェースであるＩ／Ｆ（インタフェース）回路１０１と、鍵ｅを保持するｅレジスタ１０２と、モンゴメリ変換をする乗数Ｙを保持するＹレジスタ１０３と、鍵Ｎを保持するＮレジスタ１０４と、モンゴメリ変換の演算時に行なう２Ｂ＋Ｎの値を保持するＢ２Ｎレジスタ１０５と、平文Ｘを保持するＸレジスタ１０６と、暗号化および復号化のための演算を行なう演算回路１０７と、演算結果Ｐを保持するＰレジスタ１０８と、べき乗剰余演算実行時のステートマシンとしての役割を果たすべき乗剰余制御回路１０９とを含む。
【００４７】べき乗算剰余演算回路は、さらに、モンゴメリ乗算剰余演算と剰余演算との実行時のステートマシンとしての役割を果たすモンゴメリ乗算剰余・剰余制御回路１１０と、加算および減算の演算制御を行なう加算・減算制御回路１１１と、各種モードを保持するモードレジスタ１１２と、コマンドを保持するコマンドレジスタ１１３と、ステータスを保持するステータスレジスタ１１４と、各種レジスタと演算回路１０７間でのデータのやり取りを行なうための内部バス１１５とを含む。
【００４８】べき乗剰余演算を行なうにあたり、高速化を実現するためにモンゴメリ法による乗算剰余演算を使用しているが、そのモンゴメリ法の演算［Ｐ＝ＭＲ（Ｂ・Ａ）］行なう最初に［２Ｂ＋Ｎ］なる計算を行ない、その結果をＢ２Ｎレジスタ１０５に格納する。
【００４９】モンゴメリ法の演算ＭＲ（Ｂ・Ａ）の動作について説明する。ここで、Ｂ＝Ｘ、Ａ＝Ｙの場合を考える。まず、［２Ｂ＋Ｎ］の計算を行なう。［２Ｂ＋Ｎ］の演算は、以下のようにして行なわれる。演算回路１０７は、Ｎレジスタ１０４に保持された値を０と加算し、Ｐレジスタ１０８に記憶する。演算回路１０７は、Ｘレジスタ１０６に保持された値を２倍し、その値とＰレジスタ１０８に保持された値とを加算し、Ｐレジスタ１０８に書込む。モンゴメリ乗算剰余・剰余制御回路１１０は、Ｐレジスタ１０８に保持された値をＢ２Ｎレジスタ１０５に書込む。次に、式（１９）、（２０）および（２１）の演算を５１５回繰返し実行する。
【００５０】モンゴメリ乗算剰余演算中の式（２０）を計算時、Ａmod２＝＝１およびＭ＝＝１の場合はＢ２Ｎレジスタ１０５からデータを読出し、内部バス１１５を介し読出されたデータとＰレジスタ１０８に記憶された値とを演算回路１０７にて加算する。また、Ａmod２＝＝０およびＭ＝＝１の場合はＮレジスタ１０４からデータを読出し演算回路１０７にてＰレジスタ１０８に記憶された値と加算する。Ａmod２＝＝０およびＭ＝＝０の場合は［０＋Ｐ］の加算を実行する。０は内部バス１１５を流れるデータを０にするなどして生成される。ｅレジスタ１０２は鍵ｅ、Ｙレジスタ１０３はＹ、Ｎレジスタ１０４は法Ｎ、Ｘレジスタ１０６はＸを保持するレジスタで、Ｐレジスタ１０８はモンゴメリ乗算剰余演算の式中式（１９）および式（２０）に出てくる値Ｐを保持するレジスタである。
【００５１】［演算回路１０７に設けられた加算器の構成］図２を参照して、演算回路１０７内に設けられたパイプライン処理を実行する加算器について説明する。
【００５２】この加算器は、モンゴメリ乗算剰余演算を行なう加算器の構成を複数のサブ加算器に分割し、加算器のキャリーの段数を少なくした構成としている。加算器は、サブ加算器内のキャリー回路（SubＣ回路）７２７～７４１と、サブ加算器内でキャリーが発生する可能性を示すキャリー回路（LookＣ回路）７４２～７５２と、サブ加算器間でキャリーを伝播させるためのキャリー回路（MainＣ回路）７２０および７２６と、加算結果を得るためのキャリー回路（SlaveＣ回路）７１６～７１９および７２１～７２５と、実際に加算処理を行なうアダー回路（adder）７０１～７１５とを含む。
【００５３】キャリー回路（LookＣ）７４２～７５２は、下位のサブ加算器でキャリーが発生したと仮定して、ビット同士の加算を行ない、キャリーが発生する可能性を判断する。
【００５４】一例として、１５ビットのパイプライン処理を実行する加算器について説明する。
【００５５】１５ビットの場合、下位ビットから４＋５＋６＝１５の段数が適当とする。まず、４段のサブ加算器内では普通に加算が行なわれ、キャリーが検出され、加算結果が得られる。５段のサブ加算器では、４段のサブ加算器からのキャリーを含めてSlaveＣ回路とadder回路により５段の加算結果を得る。また、別途同時に５段内だけのキャリーをSubＣ回路で検出する。またLookＣ回路は下位のビットからキャリーが発生したと仮定してキャリー信号を検出する。
【００５６】そして４段のサブ加算器からのキャリー信号で次の６段のサブ加算器へのキャリーをSubＣ回路７３５とLookＣ回路７４５からの信号どちらにするかをMainＣ回路７２０が選択する。同様に６段のサブ加算器でも加算結果を得る。
【００５７】たとえば、４段のサブ加算器において「１１１１＋１１１０」の加算が行なわれる場合を想定する。各ビットでの演算は以下のようになる。
【００５８】

０ビット	０＋１＝１	キャリーなし
１ビット	１＋１＝０	キャリーあり
２ビット	１＋１＋１（１ビットからのキャリー）＝１	キャリーあり
３ビット	１＋１＋１（２ビットからのキャリー）＝１	キャリーあり

このとき、SubC回路７３０からは「キャリーあり」の信号がMainC回路７２０に与えられる。よって、このときは、MainC回路７２０は、下位のサブ加算器からキャリーがあるとして加算演算を行なったLookC７４２～７４６のキャリーを、５段のサブ加算器のキャリーとして６段のサブ加算器に出力する。逆に、SubC回路７３０から「キャリーなし」の信号がMainC回路７２０に与えられると、MainC回路７２０は、SubC回路７３５のキャリーを出力する。

【００５９】このようにサブ加算器として分割していない場合、１４ビットの加算は０ビットからのキャリーを持って加算する必要があるので、１４段の回路分の時間が必要であるが、複数に分割すると、０ビットからのキャリーは、SubC７２７～７３０，MainC回路７２０，SlaveC回路７２１～７２５の１０段分の処理時間で済む。

【００６０】たとえば「０００１０１０１１００１１０１」と「０１０００００１１１００１０」とを加算する場合について説明する。SubC回路７２７は、０ビット目同士の加算（１＋０）を行なう。加算結果（１）はアダー回路７０１に供給される。また、キャリー（０）は、SubC回路７２８およびアダー回路７０２に供給される。アダー回路７０１は、SubC回路７２７から受取った値を０ビット目の加算結果として出力する。

【００６１】SubC回路７２８は、１ビット目の値（０および１）と、０ビット目からのキャリー（０）との加算（０＋１＋０）を行なう。加算結果（１）はアダー回路７０２に供給される。また、キャリー（０）は、SubC回路７２９およびアダー回路７０３に供給される。アダー回路７０２は、０ビット目のキャリー（０）と２ビット目の加算結果（１）とを加算して、１ビット目の加算結果（１）を出力する。同様に、アダー回路７０３～７１５では、前のビットのキャリーと着目しているビットの加算結果との加算が行なわれる。

【００６２】LookC回路７４２は、４段のサブ加算器でキャリーが発生したと仮定して４ビット目のキャリーの計算を行なう。すなわち、４ビット目の値同士の加算とキャリー（１）との加算（０＋１＋１）が実行される。演算の結果のキャリーはLookC回路７４３に与えられる。LookC回路７４７においても同様の処理が行なわれる。

【００６３】LookC回路７４３は、５ビット目の値とLookC回路７４２からのキャリーとの加算演算（０＋１＋１）を実行し、演算の結果のキャリー（１）をLookC７４４に供給する。以下、LookC回路７４４～７４６および７４８～７５２においても同様の処理が行なわれる。LookC回路７４６の出力は、MainC回路７２０に供給される。LookC回路７５２の出力は、MainC回路７２６に供給される。

【００６４】SubC回路７３１は、５ビット目同士の加算（０＋１）を行なう。加算結果（１）は、アダー回路７０５およびSlaveC回路７１６に供給される。キャリー（０）は、SubC回路７３２に供給される。SubC回路７３６でも同様の処理が行なわれる。

【００６５】SubC回路７３２は、６ビット目同士の加算と５ビット目のキャリー（０）との加算（０＋１＋０）を行なう。加算結果（１）は、アダー回路７０６およびSlaveC７１７に供給される。キャリー（０）は、SubC回路７３３に供給される。同様に、SubC回路７３３～７３５において、前のビットのキャリーと着目しているビットの値との加算が行なわれる。なお、SubC回路７３５における加算結果はアダー回路７０９のみに供給され、SubC回路７３５におけるキャリーは、MainC回路７２０に供給される。SubC回路７３６～７４１においても同様の処理が行なわれる。

【００６６】SlaveC回路７１６は、３ビット目のキャリー（０）と、４ビット目の加算結果（１）とを加算し、キャリーをアダー回路７０６およびSlaveC回路７１７に供給する。SlaveC回路７１７～７１９および７２１～７２５においても同様の処理が行なわれる。

【００６７】MainC回路７２０は、SubC回路７３０が出力する前段のサブ加算器のキャリーが１の場合には、LookCの出力をSlaveC回路７２１に供給し、キャリーが０の場合には、SubC回路７３５の出力するキャリーをSlaveC回路７２１に供給する。MainC回路７２６も同様の処理を行なう。

【００６８】以上説明したようにして、１５ビットのデータ同士の加算が行なわれる。すなわち、SubC回路、LookC回路およびMainC回路で計算が行なわれた後、SlaveC回路およびアダー回路で計算が行われる。このように二段階にわけて加算処理が実行される。

【００６９】例に挙げた１５ビットではあまり大きな段数の差は発生していないが、もっと大きなビットの加算器たとえば１３０ビットでは、分割しない場合は１２９段分の処理時間がかかるのに対し、分割した場合は３０段分の処理時間ですむ。また、６段のサブ加算器のキャリーに来る０ビットからのキャリーを求めるための回路の段数を５段（SubC回路７２７～７３０，MainC回路７２０の５段）とし、６段のサブ加算器内のキャリーを求めるための回路の段数を６段（SubC回路７３６～７４１とLookC回路７４７～７５２との２通りとも６段）としている。このような構成にすることにより、もっと大きなビットの加算器でも段数の違いがほとんどない構成にすることができる。これにより、タイミングなどに対する回路設計が容易になり、回路規模も小さくできるという効果がある。

【００７０】なお、１３０ビットのパイプラインでの加算器では、４段、５段、６段、７段、８段、９段、１０段、１１段、１２段、１３段、１４段、１５段および１６段のサブ加算器により加算器を構成するのが適当であ

る。

【００７１】［べき乗剰余演算処理について］図３は、べき乗剰余演算処理のフローチャートである。図４は、信号のタイミングを示すタイムチャートである。

【００７２】従来は、モンゴメリ法による乗算剰余演算を使用してべき乗剰余演算を行なう場合、鍵ｅのビットを検索しｅｊ＝＝１の場合は、式（１２）の［Ｙ＝ＭＲ（Ｘ・Ｙ）］を実行し、ｅｊ＝＝０の場合は、式（１２）［Ｙ＝ＭＲ（Ｘ・Ｙ）］をスキップし実行しなかった。本実施の形態では、ｅｊの値にかかわらず常に式（１２）の［ＭＲ（Ｘ・Ｙ）］を実行する。式（１２）の演算結果をＹレジスタ１０３に記憶させるときにｅｊの値３０１をチェックし、ｅｊ＝＝１の場合はＹレジスタ書込信号３０４を出してＹレジスタ１０３に演算結果を書込み、ｅｊ＝＝０の場合はＹレジスタ書込信号３０４を出さないでＹレジスタ１０３に演算結果を書込まないようする。このように、常に式（１２）を実行することにより、べき乗剰余演算時間を一定にすることができる。ＭＲ（Ｘ・Ｙ）演算信号３０２は、“Ｈ”のとき演算中であることを示す。Ｙレジスタ書込信号３０３は、“Ｈ”のときＹレジスタ１０３にデータを書込むことを示す。

【００７３】以上のように、常に式（１２）の演算を実行するようにしたため演算時間が鍵の値に関係なく一定になる。このため、タイミング攻撃に対する耐性を確保できるという効果がある。なお、タイミング攻撃とは、暗号文、鍵の長さによって、処理時間が変化することに着目し、暗号の解読を行うことを意味する。

【００７４】［べき乗剰余演算以外の演算について］べき乗剰余演算器には、べき乗剰余制御回路１０９、モンゴメリ乗算剰余・剰余制御回路１１０および加算・減算制御回路１１１が含まれる。このため、これらの制御回路を単独または組合わせて使用することにより、モンゴメリ乗算剰余演算、剰余演算、ストア演算、加算、減算、条件付加算、条件付減算など、各種演算を独立して実行することが可能である。

【００７５】このようにべき乗剰余演算を構成する各種演算を独立して実行可能にしたことによりＲＳＡ暗号以外の各種暗号の演算を行なうことができるようになる。また、何らかの原因でべき乗剰余演算器が動作しない場合に、各種演算を独立して実行することにより、動作しない原因を究明することができる。

【００７６】以上説明したように、本実施の形態では、頻繁に用いられる［２Ｂ＋Ｎ］の結果を保持するＢ２Ｎレジスタ１０５を設けることにより、モンゴメリ乗算剰余演算の高速化と演算回路の簡略化とを図ることができる。

【００７７】［実施の形態２］本発明の実施の形態２に係るべき乗剰余演算器は、実施の形態１に係るべき乗剰余演算器と同様の構成を有する。このため、その詳細な説明は、ここでは繰返さない。

【００７８】図５は、実施の形態２における信号のタイミングを示すタイムチャートである。本実施の形態では、モードレジスタ１１２に所定の値を設定することにより、べき乗剰余演算の実行方法を異ならせるものである。

【００７９】スキップモード信号４０１が“Ｈ”の場合には、ｅｊ＝＝０のときは式（１２）を演算しないで式（１３）［Ｙ＝ＭＲ（Ｙ・Ｙ）］だけを演算する。スキップモード信号４０１が”Ｌ”の場合には、実施の形態１と同様の動作をする。

【００８０】以上のように、式（１２）の演算をスキップするモードを設定できるようにしたので、テスト時間の短縮を図ることができる。

【００８１】［実施の形態３］本発明の実施の形態３に係るべき乗剰余演算器は、実施の形態１に係るべき乗剰余演算器と同様の構成を有する。このため、その詳細な説明は、ここでは繰返さない。

【００８２】図６は実施の形態３における信号のタイミングを示すタイムチャートである。通常の方法でべき乗剰余演算を行なっていくと、式（１１）の演算結果は、

$$Y=MR(1\cdot Y)=1\cdot R^{e}\cdot R^{-1}\bmod N$$
$$=R\bmod N$$

となる。最初のｊ＝ｋからｅｊ＝＝０の間、条件式のとおり式（１２）ではＹの値は変化しない。このため、式（１３）の演算だけを考えると、

$$Y=MR(Y\cdot Y)=R\bmod N\cdot R\bmod N\cdot R^{-1}\bmod N$$
$$=R\cdot R\cdot R^{-1}\bmod N$$
$$=R\bmod N$$

となり、演算結果は同じである。以上のようにｅｊ＝＝１で式（１２）［Ｙ＝ＭＲ（Ｘ・Ｙ）］を演算しＹの値が変化するまでは式（１３）［Ｙ＝ＭＲ（Ｙ・Ｙ）］を演算してもＹの値が変化しない。よってｅｊ＝＝１になるまでは式（１２）および式（１３）の演算を実行しなくても構わないことがわかる。

【００８３】通常は、演算時間を一定にするためにすべての演算を実行しているが、モードレジスタ１１２に値を設定することにより、サーチモード信号５０１を“Ｈ”にし、繰返し演算（式（１２）および式（１３））において、ｅｊ＝＝１になるまでは式（１２）および（１３）の演算を実行しないで、［ｊ－１］の演算のみ実行する。このようにｅｊ＝＝１か否かをチェックするモードを設けた。

【００８４】以上のように、式（１２）および（１３）の演算をスキップするモードを設定できるようにしたので、鍵ｅの値によってはテスト時間の短縮が大幅に図れるという効果がある。

【００８５】［実施の形態４］本発明の実施の形態４に係るべき乗剰余演算器は、実施の形態１に係るべき乗剰余演算器と同様の構成を有する。このため、その詳細な

説明は、ここでは繰返さない。

【００８６】図７は実施の形態４における信号のタイミングを示すタイムチャートである。本実施の形態では、実施の形態２で説明したスキップモードおよび実施の形態３で説明したサーチモードの両方のモードをモードレジスタ１１２に設定することができる。

【００８７】たとえば、両方のモードを設定した場合には、ｅｊ＝＝１になるまでは式（１２）および式（１３）を演算しない。一度ｅｊ＝＝１になった後は式（１２）および式（１３）をともに演算する。その後は、ｅｊの値によって、ｅｊ＝＝０のときは、式（１２）をスキップし式（１３）のみ実行する。また、ｅｊ＝＝１のときは式（１２）および式（１３）の両方の演算を実行する。

【００８８】以上説明したように、スキップモードおよびサーチモードの両方のモードをモードレジスタ１１２に設定することにより、実施の形態２または実施の形態３よりもさらにテスト時間の短縮を図ることができる。

【００８９】今回開示された実施の形態はすべての点で例示であって制限的なものではないと考えられるべきである。本発明の範囲は上記した説明ではなくて特許請求の範囲によって示され、特許請求の範囲と均等の意味および範囲内でのすべての変更が含まれることが意図される。

【００９０】

【発明の効果】本発明によると、モンゴメリ乗算剰余演算に頻繁に用いられる値をレジスタに保持することにより、モンゴメリ乗算剰余演算を高速に実行することができる。

【図面の簡単な説明】

【図１】　べき乗剰余演算回路のハードウェア構成を示すブロック図である。

【図２】　演算回路内に設けられたパイプライン処理を実行する加算器のハードウェア構成を示すブロック図である。

【図３】　べき乗剰余演算処理のフローチャートである。

【図４】　信号のタイミングを示すタイムチャートである。

【図５】　信号のタイミングを示すタイムチャートである。

【図６】　信号のタイミングを示すタイムチャートである。

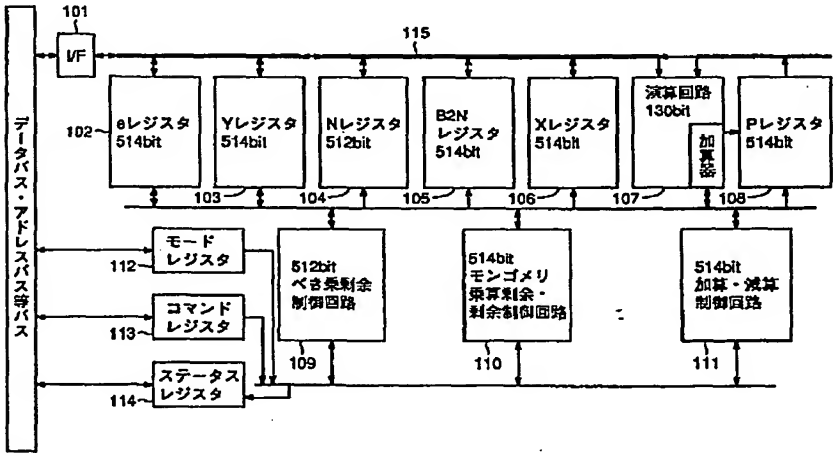
【図７】　信号のタイミングを示すタイムチャートである。

【図８】　従来のべき乗剰余演算処理のフローチャートである。

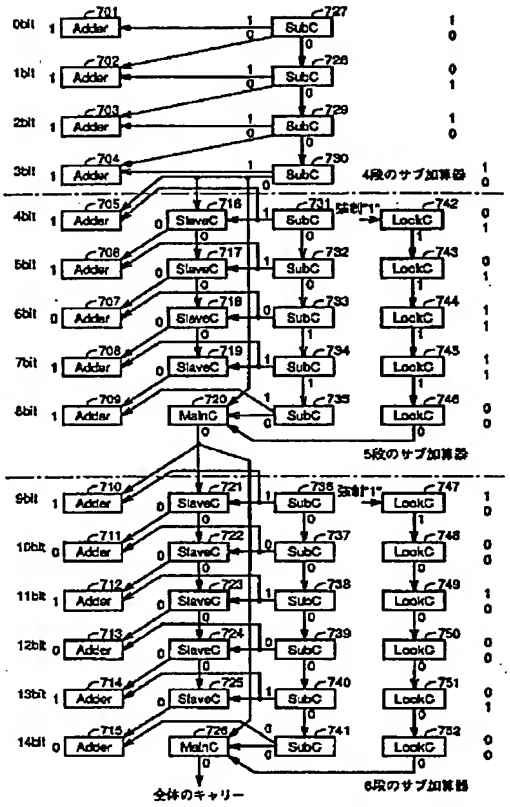
【符号の説明】

１０１　Ｉ／Ｆ回路、１０２　ｅレジスタ、１０３　Ｙレジスタ、１０４　Ｎレジスタ、１０５　Ｂ２Ｎレジスタ、１０６　Ｘレジスタ、１０７　演算回路、１０８　Ｐレジスタ、１０９　べき乗剰余制御回路、１１０　モンゴメリ乗算剰余・剰余制御回路、１１１　加算・減算制御回路、１１２　モードレジスタ、１１３　コマンドレジスタ、１１４　ステータスレジスタ、１１５　内部バス、７０１～７１５　アダー回路、７１６～７１９，７２１～７２５　ＳｌａｖｅＣ回路、７２０，７２６　ＭａｉｎＣ回路、７２７～７４１　ＳｕｂＣ回路、７４２～７５２　ＬｏｏｋＣ回路。

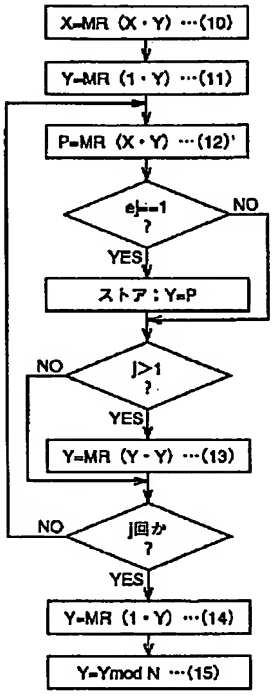
【図１】



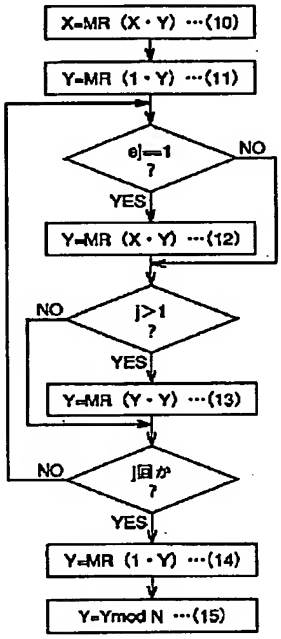
【図2】



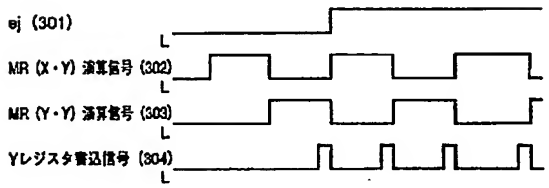
【図3】



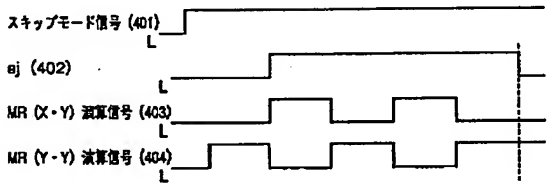
【図8】



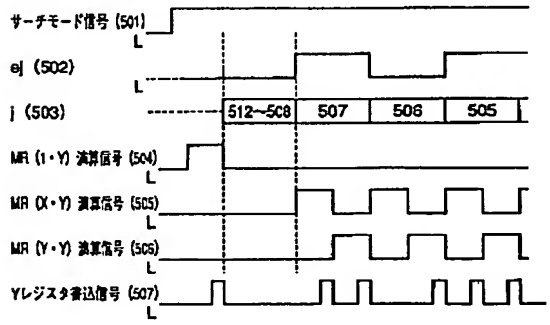
【図4】



【図5】



【図6】



【図7】